



Coreco Imaging • 7075 Place Robert-Joncas, Suite 142 • St-Laurent, Quebec, Canada • H4M 2Z2
<http://www.imaging.com>

WiT

Getting Started

Edition 8.0

Part number OC-WITM-GS000



NOTICE

© 2003 Coreco Imaging Inc. All rights reserved.

This document may not be reproduced nor transmitted in any form or by any means, either electronic or mechanical, without the express written permission of Coreco Imaging Inc. Every effort is made to ensure the information in this manual is accurate and reliable. Use of the products described herein is understood to be at the user's risk. Coreco Imaging Inc. assumes no liability whatsoever for the use of the products detailed in this document and reserves the right to make changes in specifications at any time and without notice.

Microsoft and MS-DOS are registered trademarks; Windows, Windows 95, Windows NT, and Windows XP are trademarks of Microsoft Corporation.
All other trademarks or intellectual property mentioned herein belong to their respective owners.

Printed on September 30, 2003

Document Number: OC-WITM-GS000

Printed in Canada

Contents

Introduction	1
Installation.....	3
Development	3
Location	3
Type.....	5
Standard Components.....	5
Frame Grabbers.....	6
Password.....	6
Floating License.....	7
Run Time and eWiT	9
Location	9
Type.....	9
Overview	11
Intuitive Igraphs	12
Performing a Series of Operations on an Image.....	12
Continuous Processing of Grabbed Images.....	12
Branching Data.....	13
Operating on a Set of Images.....	13
Conditional Branching.....	13
Interactive User Inputs.....	14
Dynamic Parameter Values	14
Operators and Parameters at Your Finger Tips	14
Extensive Data Inspection Tools.....	15
Nested Igraphs.....	17
Streamlined Frame Grabber Support.....	17
Parallel and Pipeline Processing.....	17
Performance Tuning Tools.....	18
User Created Operators and Data Types	19
Programming in C/C++	19
Distributed Network Processing with eWiT.....	20
Tutorial	21
Basic Usage Techniques.....	21
Displaying an Image.....	22
Starting Point of an Igraph	23
Reading and Displaying an Image.....	24
Icon Positions.....	24

<i>Creating a Link</i>	24
<i>Running an Igraph</i>	25
<i>Changing Operator Parameters</i>	25
Viewing and Studying Images	26
<i>Image Properties</i>	27
<i>Contrast and Brightness</i>	27
<i>Profiles and Magnifier</i>	27
<i>Comparing Images</i>	28
Managing Igraphs	29
<i>Saving the Current Igraph</i>	29
<i>Loading an Igraph</i>	29
<i>Clearing the Current Igraph</i>	30
Performing a Linear Series of Operations	30
<i>Building the Igraph</i>	30
<i>Running the Igraph</i>	31
Data Objects	31
<i>Data Probes</i>	32
<i>Run the Igraph</i>	32
Working with Frame Grabbers	33
<i>Frame Grabber Tool</i>	33
<i>Simple Acquisition</i>	34
<i>Continuous Acquisition and Processing</i>	34
Script Mode	35
<i>Read Image</i>	35
<i>Specify Rectangle</i>	37
<i>Select Element</i>	38
<i>Compute Statistics</i>	39
<i>Editing the Script</i>	40
Igraph Editing Techniques	41
<i>Selecting Igraph Objects</i>	41
<i>Moving Objects</i>	41
<i>Cut, Copy, Paste, and Duplicate</i>	41
<i>Editing Links</i>	42
<i>Icon Appearance</i>	42
<i>Zooming and Design Size</i>	42
Subgraphs	43
<i>Adding a New Subgraph</i>	43
<i>Editing a Subgraph</i>	44
<i>Running an Igraph with Subgraphs</i>	45
<i>Promoting Parameters from a Subgraph</i>	46
Learning More	48
Coreco Imaging Contact Information	49

Introduction

Thank you for purchasing WiT. This *Getting Started* manual will guide you through the installation process and present you with an overview of the major components of WiT.

More information about WiT can be found in the *WiT User's* manual. For users who want to create custom applications based on WiT igraps, add operators to WiT, or write stand-alone C/C++ programs using WiT processing functions, read the *Programmer's* manual. For information about WiT processing functions, read the *Operator Reference* manual.

The *WiT Help Manual* can be accessed after WiT is installed. It contains electronic versions of all the manuals, including this *Getting Started* manual, and other useful reference material.

Users who are upgrading from an earlier version of WiT are *strongly recommended* to read the *WiT Version Changes* book in the *WiT Help Manual* carefully. It describes changes in this release that require modifications to be made to existing igraps or applications.

Installation

There are three major WiT installation types:

- **Development:** This option installs the complete WiT development environment on a computer. You can use this computer to prototype designs with igraps, add new operators, create custom C/C++ programs using WiT libraries, and create custom applications with VB or VC using WiT Imaging Code (WIC), which are generated from igraps.
- **Run time:** The option installs only those components of WiT on a computer to allow a user application to run. The user application can call WiT libraries directly or execute WIC files using the WiT Engine. This option will also install eWiT for distributed processing on Intranets and the Internet.

Development

Do not attempt to install this version of WiT in an existing WiT 5.x directory. A WiT 5.x release should be uninstalled first.

You can safely install this version of WiT on top of WiT 6.x or higher.

To install the WiT development package:

1. Make sure Windows 2000 or XP is running and you have *Administrator privileges*.
2. Put the WiT CD in the CD-ROM drive.
3. Locate the hardware protection key (small gray 25-pin connector) in the WiT package. Insert this key in your parallel port.
4. Double-click the 'My Computer' icon  on the desktop.
5. Double-click the CD-ROM icon .
6. Double-click the **setup** icon .
7. Follow the on screen instructions as described below. If at any time you wish to cancel the installation, hit the Cancel button on whatever the current dialog panel is, or hit the ESC key on your keyboard.

Location

The **Choose Destination Location** dialog allows to specify where on your hard disk the WiT files should be installed. The default directory is C:\Program Files\WiT. If you want, click the **Browse** button to change the location. When you are satisfied with the location, click **Next**.



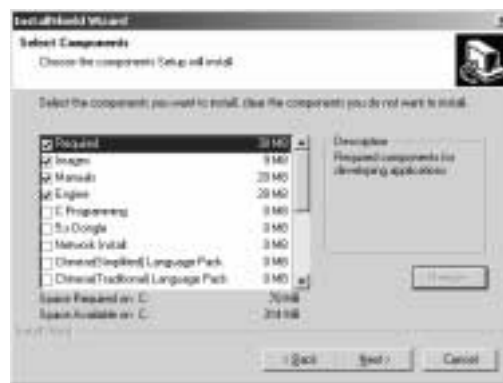
Type

Select **Development** on the **Setup Type** dialog. Then click **Next**.



Standard Components

The **Select Components** dialog allows you to select which standard components you wish to install. If you are not sure about what components you need, just use the default selections.



- **Required:** This is the core component. It includes WiT and all the processing libraries. For a new WiT installation, this component must be selected.
- **Images:** The component consists of sample images, which are useful for testing new algorithms and for running the demos. You do not have to select this component if you know you will only be using your own images.
- **Manuals:** On-line manuals: *User's Manual*, *Programmer's Manual*, etc.
- **Engine:** Files needed to execute WIC files from VB or C/C++ user applications.
- **C Programming:** Components for adding new C functions to WiT or developing C applications using WiT libraries. Includes headers, import libraries, etc.

- **5.x Dongle:** Protection key driver for old style Unikey hardware keys used in WiT 5.x and earlier versions.
- **Network Install:** This option allows you to use a copy of WiT already installed on your network to reduce the space required on your local disk.
- **Chinese (Simplified) Language Pack:** This option will install Simplified Chinese text strings and set Simplified Chinese as the default language. The required fonts must be installed separately during Windows installation.
- **Chinese (Traditional) Language Pack:** This option will install Traditional Chinese text and set Traditional Chinese as the default language. The required fonts must be installed separately during Windows installation.
- **Japanese Language Pack:** This option will install Japanese text and set Japanese as the default language. The required fonts must be installed separately during Windows installation.

Frame Grabbers

In the **Select Frame Grabber** dialog, check all the frame grabbers that you have on your PC. If you choose to install any frame grabber component, consult the manual that came with your frame grabber for hardware and driver installation procedures. All supported Coreco Imaging hardware drivers are included in the **\Drivers** directory on the WiT CD-ROM. If you plan to use a USB web cam or DV camera with IEEE 1394 (firewire), you need to install Microsoft DirectX, which is included in the **\redist\directx8** directory on the WiT CD-ROM. You must also follow the installation procedures that comes with your web cam or DV camera. Click **Next** to continue.



Password

After **setup** is complete, and before you run WiT, enter your 20-digit password numbers using the WiT License Manager  and the license file diskette provided in your WiT package. A password is used to authenticate various software products you have purchased and enables the products to run correctly on your machine. Start the License Manager from the WiT program group. The License Manager presents a product listing as shown below:

Product	Status	Support	Licenses
WiT	Key Detected	2004/09	1
WiT 2.0/2004	No Password	1/01	0
WiT 2.0/2005	No Password	1/01	0
WiT 2.0/2006	No Password	1/01	0
WiT 2.0/2007	No Password	1/01	0
WiT 2.0/2008	No Password	1/01	0
WiT 2.0/2009	No Password	1/01	0
WiT 2.0/2010	No Password	1/01	0

Select the **Load passwords...** item under the **File** menu and load the license file ('.lic' file) from your license diskette. After the file is loaded, your product list information should be updated with the serial number, key type, expiry date, and number of licenses of each purchased product. In addition, the status should change to "Key Detected".

Alternatively, passwords can be entered by clicking on a product from the product list then selecting the **Set password...** menu item under the **File** menu. Enter the product password in the text entry box then click the **OK** button. If you have typed in the password correctly then updated information should appear in the product list. Once your WiT password is entered, you will be able to start WiT and begin work.

The **Support** date refers to the date at which your one year of *free upgrade and technical support* ends. Your WiT license itself will never expire. So you can use the version of WiT you purchased for as long as you want. However, for keeping up with the latest improvements and receive technical support, we recommend that you keep your software licenses up-to-date by renewing your support period every year.

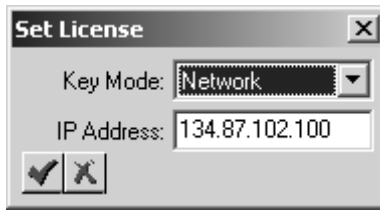
Floating License

WiT is copy protected by a hardware key and passwords. By default, the hardware key is provided by a dongle connected to the parallel port of the computer used to run WiT.

When multiple users need to share one or more copies of WiT, a floating license arrangement can be used. A floating license allows WiT to run on several networked PCs with only one hardware key. When using floating licenses, dongles are typically installed on a file server. This can help reduce the possibility of dongles being stolen.

To enable floating license operation, follow the procedure below:

1. Start the License Manager and select the **Floating license...** menu item under the **File** menu. This action pops up a license dialog panel as shown below.



2. The **Key Mode** determines whether a license should be obtained by a local hardware key or via a floating license. Select the **Network** option.
3. Now you must pick a machine to act as the floating license host and ensure it is on your network with a hardware key installed. Start the WiTRAS program  on the floating license host machine. Note the IP address reported in the title bar of WiTRAS, e.g. 134.87.102.69.



4. Enter the floating license host IP address you obtained in step 4 into the **IP Address** field. Click OK on the dialog panel and exit the License Manager.
5. The next time you start WiT, a connection to WiTRAS running on the floating license host will be made to check out a license. WiTRAS verifies the number of copies allowed and accepts or rejects a request for a license. If you have a number of products with varying numbers of licenses that are managed by WiTRAS, the first user who connects for a license will receive all of the products. As the number of users requesting a license increases, the product mix will decrease in functionality based on the remaining number of licenses for the various products.

Run Time and eWiT

A WiT run time installation installs components which are necessary to execute user programs created in C/C++ or VB which call the WiT libraries directly or call WIC files executed by the WiT Engine, or for distributing computation over several networked computers with eWiT.

To install a WiT run time:

1. Make sure Windows 2000 or XP is running and you have *Administrator privileges*.
2. Put the WiT CD in the CD-ROM drive.
3. Locate the hardware protection key (small gray 25-pin connector) in the WiT package. Insert this key in your parallel port.
4. Double-click the 'My Computer' icon  on the desktop.
5. Double-click the CD-ROM icon .
6. Double-click the **setup** icon .
7. Follow the on screen instructions as described below. If at any time you wish to cancel the installation, hit the Cancel button on whatever the current dialog panel is, or hit the ESC key on your keyboard.

Location

The **Choose Destination Location** dialog allows to specify where on your hard disk the WiT files should be installed. The default directory is C:\Program Files\WiT. If you want, click the **Browse** button to change the location. When you are satisfied with the location, click **Next**.



Type

Select **Runtime** in the **Setup Type** dialog:



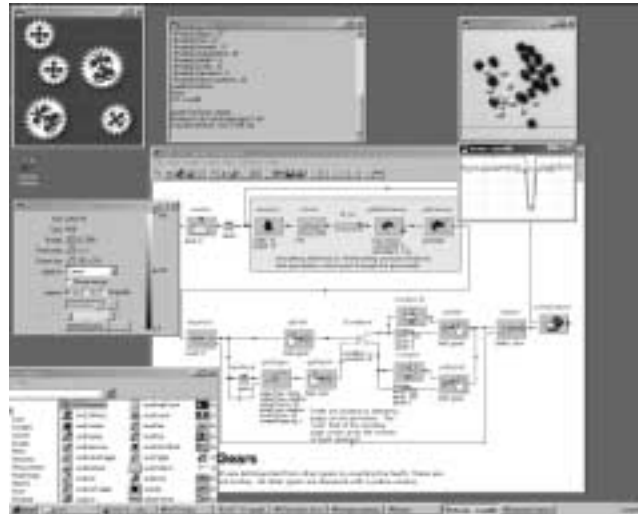
Then install other components by following the same procedures as the Development installation.

Overview

WiT is a powerful visual programming tool for designing computer algorithms with block diagrams, called igraps, which can be *executed*. Igraps are built using icons and links with point-and-click simplicity. They are particularly suitable for complex image processing applications in areas such as machine vision, biomedical imaging, and imaging research.

An igrap presents an entire algorithm using proven building blocks (icons) to perform image processing. Links direct data from one icon to the next and can be configured to enable probes or breakpoints. Links can be split to deliver the same data to many blocks. Parameters for operators are easily changed with quick pop-up dialog panels. All the standard features of a CAD-like environment such as cut, copy, paste, undo, zoom, area select, grid snapping, and pan are supplied for easy igrap design.

The following figure shows an example igrap and a typical WiT session:



Flow control is handled by special operators to perform looping, if-then-else, sequencing of large data sets, collecting, I/O, counting, and general data manipulation. Igraps can be nested by creating operators which encapsulate an entire igrap for use in other igraps. This hides unnecessary detail and promotes reuse of common processing steps.

There are hundreds of processing functions that comes standard with WiT. Users can also add their own C functions and use them iconically as efficiently as any built-in WiT operator.

Igraphs designed in WiT can be converted into a hidden code executable by user applications written in C/C++ or VB. WiT can also generate C code from igraps to be incorporated into user C programs. Igraphs can exploit multiple CPUs and networked PCs or Coreco Imaging Mamba boards to give you unprecedented parallel processing capabilities. All these features are at your fingertips when you use WiT.

Intuitive Igraphs

The most distinctive feature of WiT is its use of imaging graphs, or *igraps*, to represent algorithms. E.g.:

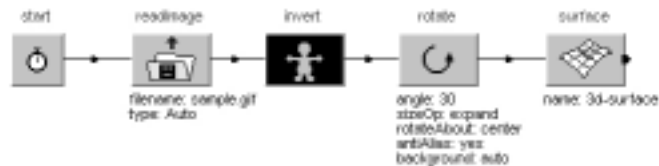


Igraphs are like block diagrams. They are much more intuitive and less restrictive than traditional linear textual programming code. Because igraps use links to connect data paths between operators, variable names are not required. This makes experimental development of new algorithms much easier because you can change operators and execution sequences without having to rename data variables.

Following are some more interesting example igraps.

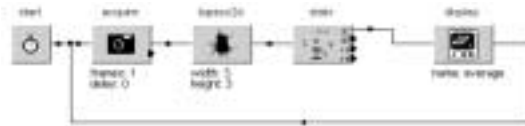
Performing a Series of Operations on an Image

This igrap reads an image from a file and applies a series of operations on the data. Contrast the simplicity and the absence of variable names of this igrap with a similar algorithm written in C or BASIC.



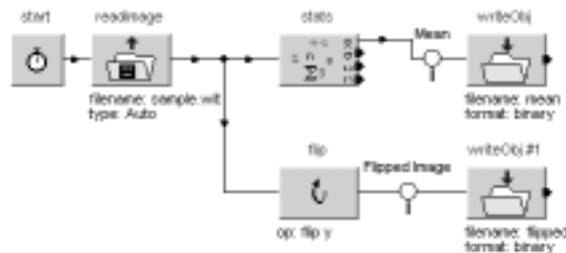
Continuous Processing of Grabbed Images

This igrap applies a series of operations on a continuous stream of images acquired from a frame grabber. This is typical of most machine vision applications.



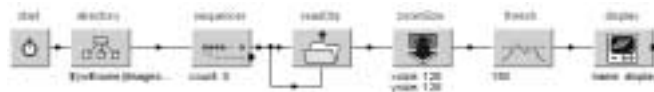
Branching Data

This iGraph branches off a data object to be processed by different operators. This is more intuitive than the artificial linear restriction imposed by traditionally textual programming languages. In addition to looking more natural, such branches are key to WiT's ability to utilize parallel CPUs and networked computing resources to accelerate an algorithm.



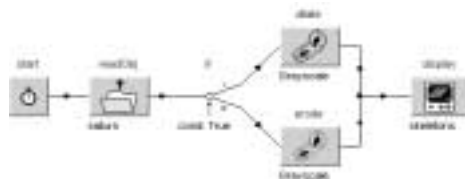
Operating on a Set of Images

This iGraph applies the same operations to a set of images. This kind of batch processing is difficult or unsupported by most 'easy-to-use' machine vision or graphics programs. Not only can WiT iGraphs perform such tasks, it does so with sheer simplicity.



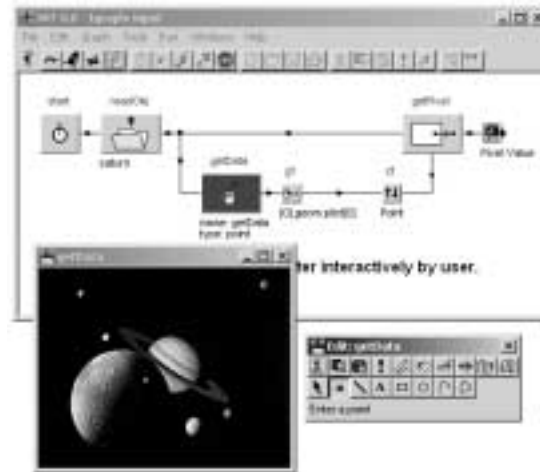
Conditional Branching

This iGraph illustrates how conditional branching can be accomplished.



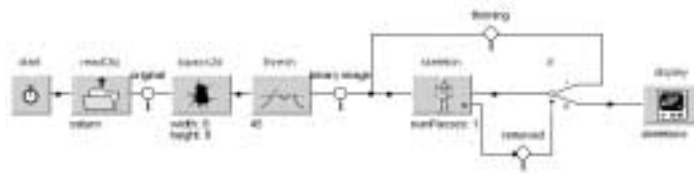
Interactive User Inputs

This igrph pops up a image for the user to enter a location on an image and then reports the pixel value at that location. It illustrates how an igrph can be paused to allow a user to enter data.



Dynamic Parameter Values

This igrph shows that parameters, which are normally fixed at design time, can be converted into inputs and fed by other parts of an igrph dynamically. In this example the **condition** parameter of an **if** operator is calculated dynamically to perform a finite operation loop. This is of course normally how **if** operators are used since a statically programmed branch (as illustrated by the Conditional Branching example) is not a usually a useful construct.

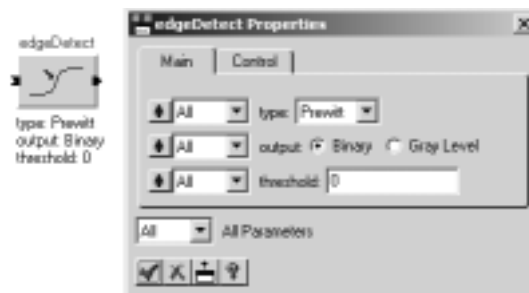


Operators and Parameters at Your Finger Tips

Instead of flipping through big reference manuals looking for the right function, WiT presents all its operators graphically. You can view them as functional groups, sort them alphabetically, use wild card filters, etc. And on-line help for an operator is just a click away.

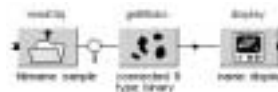


WiT presents all parameters graphically, with default values that usually work well, and value ranges that are enforced to be valid.



Extensive Data Inspection Tools

You can look at data values using a **display** operator or a probe () on a link:



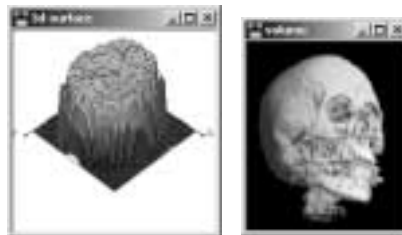
All kinds of images can be displayed: 8-bit, 16-bit, signed, unsigned, floating-point, complex, etc. Image displays have profiles and pixel value magnifiers that help you analyse data values:



Non-image data of any kind can be displayed as plain or formatted text, with all kinds of user settings for font and color:

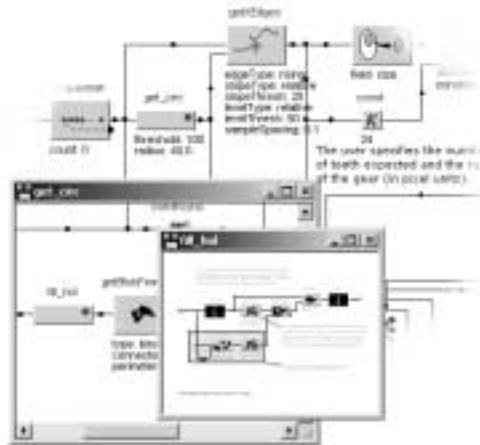


You can even display images as surface plots and view 3-D volumes:



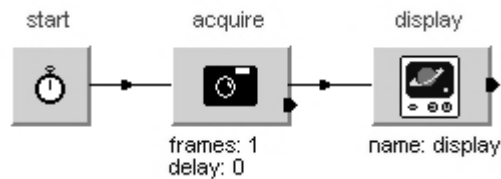
Nested Igraphs

Portions of an igraph can be displayed as a single icon to clarify the design of an igraph. Such icons represent sub-graphs. Sub-graphs can be nested as deep as you want.



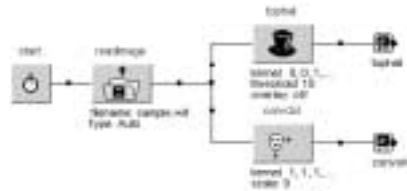
Streamlined Frame Grabber Support

A uniform interface is used to control all frame grabber types. For example, once a frame grabber is correctly installed, the **acquire** operator is used to acquire images. There are no special parameters to set.

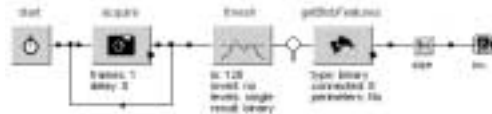


Parallel and Pipeline Processing

Graphs are inherently more suitable for expressing parallelism. For example, in the following graph, WiT will execute both branches simultaneously if the PC has multiple CPUs or if other distributed processors are available.



In addition to parallel branches, WiT can also run operators in parallel when data is pumped into a pipeline. In the following example, WiT will keep up to five processors busy when the pipeline is filled.

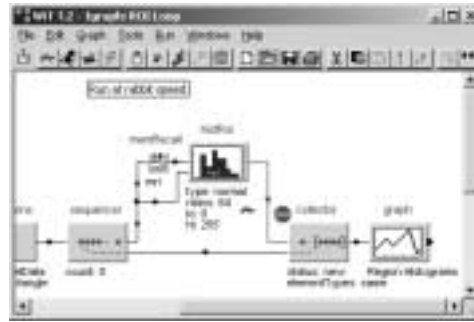


Performance Tuning Tools

WiT is designed for high-speed machine vision. WiT has an integrated profiler to locate bottlenecks in an igrph:

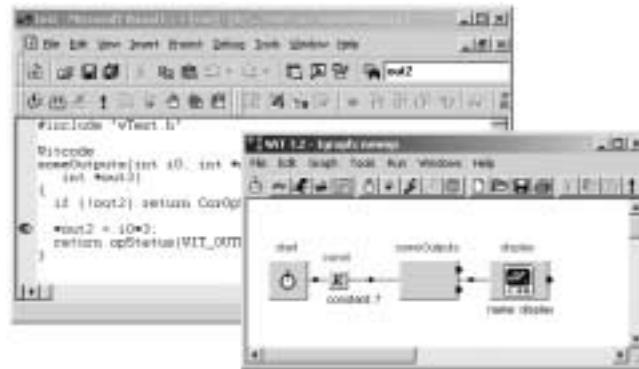
Operator	#	Time [ms]	Min [ms]	Max [ms]	
convert	1	18	2.788	0.075	0.458
get_imagefield	2	30	3.500	0.006	0.623
sequence	1	18	3.517	0.198	0.303
get_imagefield...	1	18	3.504	0.137	0.398
get_imagefield...	1	18	4.000	0.171	0.353
get_imagefield...	1	18	5.436	0.078	2.009
get_imagefield...	2	30	6.813	0.136	0.398
get_imagefield...	1	18	7.893	0.417	0.680
get_imagefield...	1	18	15.504	0.004	1.658
get_imagefield...	1	18	16.612	0.736	2.015
sequence	1	18	18.993	0.680	5.967
sequence	2	18	29.778	0.680	8.148
get_imagefield...	2	30	60.123	1.355	5.636
readData	1	1	300.006	---	---
get_imagefield...	1	18	296.799	8.713	13.322
writeData	2	18	347.036	17.948	91.387
Total	38	408	817.128		

WiT also has different run speeds to control the amount of execution feedback and break points to debug an igrph or to run it at maximum speed.



User Created Operators and Data Types

You can add your own functions and use them just like any WiT operators in your igraps. WiT can load a library project into Microsoft Visual Studio for C/C++ directly. Then you can compile and debug your operators. You can even add your own icons and on-line help to share your custom operators with your colleagues.



Programming in C/C++

All WiT operators are implemented as C functions in DLLs. So if you are more comfortable with writing your own C/C++ programs, you can use the WiT libraries for image processing and analysis directly from your C/C++ program. The programming API for WiT operator functions are very natural for C programming and are fully documented. You can use WiT for prototyping, then use the C code generator to generate code that you can insert into your C/C++ application. You can also bypass the entire WiT and igraps concept and code directly in C right from the beginning. The following is a trivial yet complete example console application using WiT libraries:

```
#include "corObj.h"
#include "wSystem.h"
```

```

main(void)
{
    CorImage im;

    CorObjInit(NULL, NULL);
    if (cor_rdImage(&im, "../data/sample2.bmp", 0) != COR_OP_OK) {
        printf("Failed\n");
    } else {
        int x, y;
        CorUByte min, max;
        int w = CorObj_width(&im);
        int h = CorObj_height(&im);
        CorUByte *ip = CorObj_mdData(&im);

        min = max = *ip;
        for (y=0; y<h; ++y) {
            for (x=0; x<w; ++x) {
                if (*ip < min) min = *ip;
                if (*ip > max) max = *ip;
                ++ip;
            }
        }
        printf("Size: %dx%d, range: %d-%d\n",
            w, h, min, max);
        CorImageFree(&im);
    }
    CorObjExit();
}

```

Distributed Network Processing with eWiT

WiT supports parallel execution on distributed computers over a TCP/IP network, by connecting WiT (the client) to a number of eWiT servers. eWiT (for Embedded WiT) is a program that supports most of the capabilities of WiT, with the exception of the graphical user interface. It communicates with WiT using TCP/IP when connected via a network. eWiT can execute subgraphs when instructed by the WiT client. Because it has most of the capabilities of WiT, eWiT can be configured (using the WiT Manager) to support all kinds of frame grabbers and hardware, and provide those services to WiT remotely. WiT coordinates all the local CPUs and eWiT connections so that many operators in an igrph can be executed at the same time. You can make as many eWiT connections to WiT as you want. A computer that runs eWiT must have at least a WiT run time license. A WiT development license will also enable eWiT.

There are various reasons for using eWiT. If your computer is connected to a local area network, you may want to run eWiT on all machines that are not busy, so that your igrphs can execute faster. Usually communication speed is much slower compared to execution speed. Therefore, in general, you only want to distribute subgraphs that take a fairly long time to execute to eWiT. Another common reason for using eWiT is for grabbing images through networked computers where the frame grabber is on one computer but the igrph and developer are on another computer elsewhere.

Tutorial

WiT is a graphical programming environment that allows you to develop and test imaging algorithms quickly using igraps (imaging graphs). WiT looks like this when it starts up:



WiT on Start Up

The main WiT window consists of three areas: the menu bar, the tool bar, and a large workspace on the bottom. The smaller window on top is a status window for displaying messages.

Basic Usage Techniques

As with many complex tools, most people prefer to learn by experimentation rather than going through thick manuals. This tutorial presents step-by-step instructions which should be easy to follow and not take too long to go through. However, if you want to skip even this and start experimenting right away, please read this section at least!

Like all graphical programs, most of your interaction with WiT is done via your mouse. WiT can work with either a two button mouse or a three button mouse. *If you have a three button mouse, you can use the middle mouse button to perform the same function as the left mouse with the **control** key held down.* Specifically, WiT recognizes these mouse actions for various functions:

Button	Down	Up	Drag
Left	Select Start area select	Done area select	Move and modify objects
Left with control key down	Add to or remove objects from selected list	Backup when entering polyline	
Right	Display pop-up menus or property dialog		

Context sensitive on-line help is available for operators. Point the mouse at the operator icon and hit the **F1** key. This manual is available on-line from the **Help/Contents** menu item.

Throughout the documentation and when running WiT, **\${WITHHOME}** stands for the installation directory in which you have installed WiT, which is **C:\Program Files\WiT** by default. **\${WITUSR}** refers to the directory where your personal copy of WiT resides. In a single user installation, **\${WITHHOME}** is usually the same as **\${WITUSR}**.

With this knowledge, you can skip reading manuals and begin to explore WiT!

Displaying an Image

Let us start our tour with a simple task: read an image from a file and display it. Many people like to describe tasks and algorithms with block diagrams because they are visual and easier to understand than a textual description. It would be nice if the same block diagram can actually be used for the *execution* of the algorithm. This is precisely what WiT lets you do: it allows you to specify an algorithm with a graph (block diagram), called an *igraph* (for imaging graph). Igraphs are made up of icons (blocks) connected with links. Icons represent operators, e.g. read an image, perform a convolution, etc. Links specify paths along which data travels from one operator to the next. The order in which operators are executed is determined by the flow of data. We will build the following igraph:



Igraph for Reading and Displaying an Image

You will see that not only is the igragh a concise and visually pleasing representation of the algorithm, but it is also much easier to create and run than an ordinary computer program.

Starting Point of an Igragh

First we need an operator to start an igragh. Pull down the **Tools** menu and select the first item: **Operator Explorer**. The Operator Explorer window comes up:



Operator Explorer

The Operator Explorer is designed to look and work like the Windows Explorer. On the left is a list of library names, grouped according to function. Click the left mouse button on a library to make it active. On the right is the list of operators belonging to the active library. The first library group is always **All**, selecting it will give you all the operators currently available. When the Explorer first comes up, the active library is **All**. For all the exercises in this chapter, we will use the Explorer with the **All** selection.

The first operator we need is **start**, which is the starting point of the flow of an igragh. Locate the **start** icon on the right side of the Explorer. You will need to use the horizontal scroll bar on the bottom to bring the icon into view. Press and hold the left mouse button at the **start** icon, then drag it to the igragh workspace (the large blank area in the main WiT window). When your mouse moves to the workspace area, a copy of the **start** icon will follow your cursor. Move somewhere to the left of the workspace, then release the left mouse button. This creates an instance of the **start** operator.

Each operator has its own unique properties. Most operator properties have reasonable default values, which will work fine for this tutorial. We will look at how to change parameter values later.

Reading and Displaying an Image

Next we need an operator to read an image. Choose **readImage** from the **Explorer** and follow the same procedures to place the icon on the igragh. Put it somewhere to the right of **start**. Note that operator properties are shown below the operator icon. Using the same procedure, put a **display** operator somewhere to the right of **readImage**. You should now have something like the following figure:



Igragh after readImage and display are Instantiated

Icon Positions

If you are not happy with the positions of **start**, **readImage** or **display** on the igragh, point your mouse at the operator icon you wish to move, then press and **hold** the left mouse button. Now as you *drag* your mouse, the operator follows. The operator is placed wherever you release the mouse button.

Creating a Link

Notice the black arrow head on the right of **start**, and the black arrow tail on the left of **readImage**. The arrow head symbolizes an output port: data going out of the operator. The arrow tail symbolizes an input port: data coming into the operator. Input ports usually appear on the left side of operator icons, output ports on the right.

We now need to connect the two icons together with the proper data-flow. Point at the output port of **start** and click the left mouse button. Now when you move your mouse, a rubber band wire is created between the port and the cursor. Move the cursor to the input port of **readImage** and click the left mouse button. The two ports disappear. A link with an arrow indicating the direction of data-flow connects the output of **start** to the input of **readImage**.

Repeat the process to connect the output port of **readImage** to the input port of **display**. You should have something like the following figure:



Igragh after Connecting Input and Output Ports

Running an Igraph

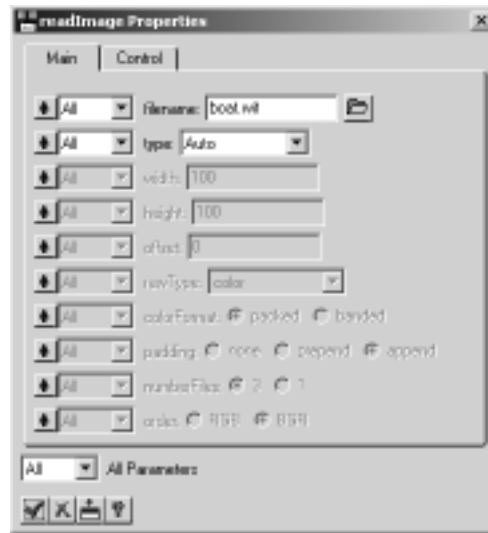
Click the  on the tool bar. The status indicator on the left of the tool bar turns from the static  icon (a mouse, symbolizes editing) to an animated  icon. **Start** turns green for about a second, and a message comes up on the WiT status window stating that **start** is now executing. A red dot (representing the data object) then travels from the output of **start** to the input of **readImage** along the link. The **readImage** operator executes, producing an image at its output port represented as a new red dot that travels to the **display** operator.

When the data arrives at **display**, **display** turns green for a second, and a new window pops up, with the image (**sample**) read by **readImage** shown. The name of the image window is **display**, which is copied from the **Name** parameter on the **display** operator. WiT reports on the status window that the igrph execution is complete, and the status indicator on the tool bar changes back to .



Changing Operator Parameters

Now let us display a different image by change a parameter. Point your mouse at **readImage**, and click the right mouse button. The operator properties panel for **readImage** will come up.



readImage Operator Properties

Change the **filename** param to *boat.wit*. Leave all other parameters at their default values. Click the  button. Run the igrph again by clicking the . You should see the following image:



Boat Image

To follow the rest of this tutorial, change **filename** back to *sample.wit* and run the igrph again. Leave the image window up.

Viewing and Studying Images

After running the previous exercise, you will see an image displayed. Images are initially shown at their original size when it first comes up, but you can resize it using the resize corners. WiT has many features for you to look at and compare pixel values critically.

Image Properties

A number of tools are available for studying an image while it is displayed. Press and hold the *right* mouse button inside the **sample** image. A pop-up image menu appears. Select **Properties**. An image properties window comes up:

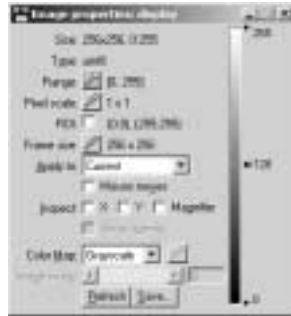


Image Properties Panel

Contrast and Brightness

On the right of the image property panel is a grayscale ramp with three arrows. You can control the contrast of the image by dragging the arrows. For example, press and hold the left mouse button at the top arrow, and drag the arrow down to around 128, for example. The image becomes brighter. Dragging the middle arrow shifts all three arrows without changing the separation between the arrows. Experiment with all three arrows, and observe the effects.

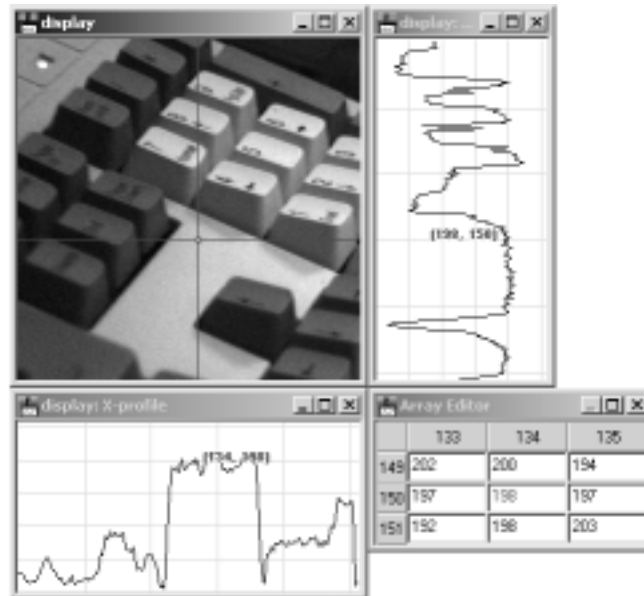


Brightness and Contrast Adjustment

Profiles and Magnifier

Check the **Magnifier** box for **Inspect**. A new window comes up showing you the actual pixel values and cursor position. Now try dragging (press and hold) the mouse with the left mouse button down

inside the image window. You will see the pixel position and value displayed interactively on the image property panel. Next check the **X** and **Y** boxes for **Inspect**. Two more windows come up, one for a plot of all pixel values along the x-axis (horizontal line), the other of pixels along the y-axis (vertical line). Again, as you drag your mouse in the image window, the profile plots are updated interactively.



Inspecting Pixel Values of an Image

Comparing Images

WiT has powerful features which allow you compare images critically. It also allows you to view images stored as files directly without running an igrph. To try this, select **File/View...** from the menu bar. A file dialog comes up. WiT comes with many sample images. Go to the directory **images** sub-directory under the WiT installation directory. For example, if you have installed WiT in 'C:\Program Files\WiT', then navigate to C:\Program Files\WiT\images) in the file dialog. You should now see a large number of image files listed.

To best illustrate the usefulness of the image comparison features in WiT, choose an image which is the same size and type as the **sample** image which should be already displayed at this time. **Sample** (now displayed with name **display**) is a 256×256 8-bit unsigned image. **Sample2** is also a 256×256 8-bit unsigned image, so choose it from the file dialog. The image comes up next to **sample** in another window.

When multiple images are displayed, clicking anywhere on an image causes the image property panel to switch focus to that image. You can tell which image has the focus by looking at the title bar on the image property panel: the name of the active image is shown there.

On the image property panel, choose **All** for the **Apply to** item. Now all images will try to do the same thing that you do to one image. For example, turn on the X-profile plot for **sample2**. Now if you drag your mouse in **sample**, both the X-profile plots for **sample** and **sample2** are updated in unison. Uncheck the X-profile check box on the property panel, and both X-profile plots are closed. Try other controls, such as the ramp or colormap, to see how powerful the **Apply to** control is.



Comparing Images

Managing Igraphs

Igraphs are stored as files, you can manipulate them like most files.

Saving the Current Igraph

To save the igrph you are editing, click the  tool bar button. A file dialog comes up. Enter an appropriate name for the igrph, say 'mygraph', then click the **OK** button.

Loading an Igraph

To load an igrph, click the  tool bar button. A file dialog comes up. All the igrphs described in this section have already been pre-built and tested. They are stored in the **demo\Eg** sub-directory. For example, to see what the pre-built **Displaying an Image** igrph looks like, enter the name **Displaying an Image** for the file name, and click the **OK** button. The current igrph is cleared, and the new igrph is then displayed in the workspace.

Clearing the Current Igraph

To clear the current igragh, click the  tool bar button. The workspace is cleared, and all pop-up data windows (the **sample** image window and its properties) are removed.

Performing a Linear Series of Operations

The power of igragh lies in the ability to create and test complex algorithms quickly. In our next exercise, we will read an image, convert it to its negative, rotate it 30 degrees, then display the result as a 3-D plot.

Building the Igraph

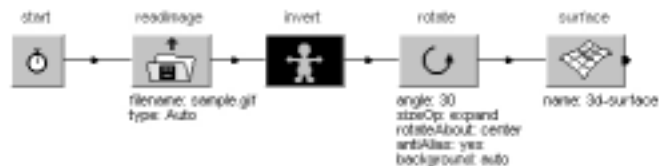
If you do not wish to go through the editing process, load the pre-built igragh **Performing a Linear Series of Operations**, and skip the following building instructions.

To build this igragh, first clear the workspace: click the  tool bar button. Create an instance of **start**. Put it somewhere on the left of the workspace.

Next create an instance of **readImage**. Place it to the right of **start**. We will use the same **sample** image, so accept all the default parameters by clicking the  button.

Next create an instance of **invert**. Place it to the right of **rdImage**. **Invert** has no parameters, so just click . Connect the output of **readImage** to the input of **invert** by creating a link. Create an instance of **rotate**. Set the **angle** parameter to 30. Connect the output of **invert** to **rotate**. Create an instance of **surface**. Use all the default properties. Connect the output of **rotate** to **surface**.

You should have the igragh as follows:



A Series of Operators

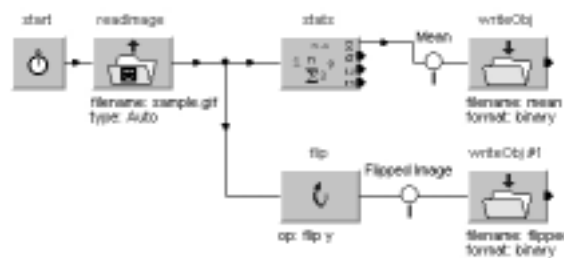
Running the Igraph

Click the  button. The operators will each turn green for a second, in the order specified by the links. When the token reaches **surface**, a window pops up with a 3-D plot of the image. The pixel intensity is plotted on the Z-axis.

Data Objects

Data objects in WiT are represented internally by a special format. This format allows WiT operators to check whether the object it receives is the correct type. For example, if an operator expects an image as input, and you send it an integer, the operator can issue a warning and refuse to process the data. Some operators can deal with a variety of input types, doing different things depending on the data type. This data format is preserved when a WiT data object is saved to a file, so that when a data object is read back, WiT will know what data type it is. It also means that only one operator (**readObj**) is required to read a data object from a file, and only one operator (**writeObj**) is required to write a data object to a file.

In this exercise, we will explore how WiT handles different data types with this igragh:



Igraph for Demonstrating Use of Data Objects

This igragh introduces some new igragh features:

1. Some links (between **stats** and **writeObj**, and between **readImage** and **flip**) are not simple straight lines, but have corners.
2. The link from **readImage** branches off into two links.
3. There are two instances of the same operator **writeObj**, and one of them has a **#1** attached to its name.
4. There are magnifying glasses on the links between **stats** and **writeObj** and between **flip** and **writeObj #1**.

Because of these new features, it is worthwhile for you to go through the editing process for this igragh, although a pre-built version does exist (**demo\Eg\Data Objects.igr**).

First create one instance of **start**, **readImage**, **stats**, and **flip**, and place them at the appropriate positions. First connect the output of **start** to the input of **readImage**. To create the link branches, first connect the output of **readImage** to the input of **stats** as usual. Now point the mouse at the middle of this link and double-click the left mouse button. A circular dot appears at the position where you clicked your mouse. This is a newly created link *junction*. As you move your mouse, a rubber band link exists between the mouse cursor and the junction. Click the left mouse button once to create a link corner, then point at the input port of **flip** and double-click the left mouse button. The link branches are now properly created.

Next make an instance of **writeObj**. Set the **filename** parameter to **mean**. Connect the first output (the mean) of **stats** to the input of **writeObj**. Leave the other outputs of **stats** unconnected. Make another instance of **writeObj**. Set the **filename** parameter to **flipped**. Place it to the right of **flip**. Notice the name is automatically set to **writeObj #1**. WiT does this because all objects (operators and links) in an igraph must have unique names. On the property panel, you can change the name of the operator to almost anything you like, provided it does not conflict with any other objects in the igraph. If the name you type is already in use, WiT will append a number to it. Connect the output of **flip** to **writeObj #1**.

Data Probes

The only remaining components now are the magnifying glasses . The magnifying glass (or probe) allows you to inspect a data token without having to invoke the **display** operator. A probe can be placed anywhere on a link. When a data token travels on that link, a pop-up window will be created to display the contents of that token. To create a probe on the link between **stats** and **writeObj**, click the **right** mouse button at the position you want the probe to be placed. A property panel for the link comes up:



Adding a Probe

Check the box for **probe**. Notice the link has a name, which starts with an underscore ('_'). Igraph object names that start with an underscore are not displayed. This feature is used to avoid displaying meaningless names which can clutter the appearance of an igraph, particularly so for links. However, when we place a probe on a link, the name of the link will be used as the name of the pop-up data window later, so it is helpful to give the link a meaningful name. Name the link **Mean**. Repeat the process to add a probe for the link between **flip** and **writeObj**. Name the link **Flipped Image**.

Run the Igraph

Now run this igraph by clicking the  button. As the token from **readImage** reaches the link junction, it splits into two and the new tokens travel simultaneously to the input ports of **stats** and **flip**. Later you

will find that this branching action enables WiT to make use of multiple computers to speed up execution of igraps. As the tokens travel under the magnifying glasses, two pop-up windows appears:



Data Objects

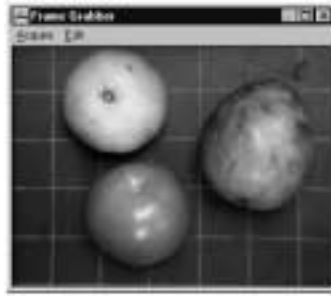
One is named **Flipped Image**, and contains a flipped image of **sample**. The other is named **Mean**, but this one contains a single number, which is the mean of all the pixel values in the **sample** image. This number is also a WiT data object. A WiT data object can be simple, such as a single integer or character string, or it may be complex, consisting of nested data structures and arrays. Images are in fact just a particular type of data object. All data objects other than images are displayed as formatted text. These two data objects are saved to files by **writeObj**. Notice you do not need to use different operators for different object types. The **writeObj** operator handles *all* WiT data types.

Working with Frame Grabbers

If you have purchased a frame grabber for use with WiT, you can try this exercise. Otherwise, skip this section.

Frame Grabber Tool

If you have a frame grabber board supported by WiT, then start WiT with a configuration file that loads the particular frame grabber. This is done by clicking the Windows **Start** button and selecting **Programs/WiT/Start**, and selecting the frame grabber you are using, e.g. Bandit. This causes WiT to load a configuration file that includes support for the selected frame grabber board. Bring up the **Tools** menu. If the frame grabber board was installed correctly and is operational, then you will see that the **Frame Grabber** item is enabled on this menu. Select this item to bring up a frame grabber window which allows you to display live video, grab individual frames and control frame grabber parameters such as grab size and input channel selection. This tool is a convenient way of making sure your frame grabber is working. To display live video, select **Live** from the **Acquire** menu. Select **Acquire** again to stop acquiring. If everything is working then exit the **Frame Grabber** tool and select **Save config** under the **Tools** menu so the next time WiT is started, your frame grabber will be automatically ready to use. For most uses, this is all you need to know. For sophisticated uses of frame grabbers, such as triggered acquisition and non-standard cameras, the Frame Grabber Tool provides many controls.



Frame Grabber Window

Simple Acquisition

Next, let us use your frame grabber in an igrph. If you have followed the previous section, then WiT would have come up with everything correctly configured to use the frame grabber. In this exercise, we will simply grab an image and display it.

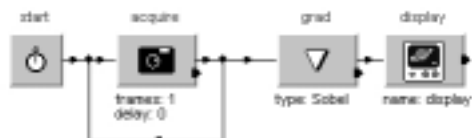


Igrph for Grabbing an Image

Build this igrph yourself or load it from the examples directory. Run it and you will see an image grabbed from your camera! You can insert some processing between **acquire** and **display**. Depending on the capabilities of your frame grabber, some or all WiT processing operators may run on the hardware.

Continuous Acquisition and Processing

To do continuous processing, modify the igrph as follows:



Continuous Processing

When you run this igrph, you will get a continuous display of edge filtered images from your camera. To stop the igrph, click the  button.

Script Mode

Although igraphs are extremely powerful and easy to use, there are times when you just want to apply one or two operators to some data objects. For example, you may want to read an image and transpose it, then save it back. Having to build a tiny igraph and then run it is inconvenient, especially when you know you will never use the igraph again. So WiT provides a script execution mode. In this mode, operators that you select are executed and the results shown immediately. Script mode is useful in exploring and understanding the rich set of imaging operations WiT offers, as well as debugging any new operators that you have developed yourself. You can switch to script mode by clicking . To go back to graph mode, click .



WiT in Script Mode

The menu bar and tool bar remains the same in script mode, although some items may be disabled. The workspace is used to display thumbnails of data objects. Operators are invoked just as in graph mode. If an operator requires inputs, they are retrieved from the workspace. After execution, any outputs an operator produces are displayed in the workspace.

In this exercise, we will compute the average pixel value of a region within an image in script mode.

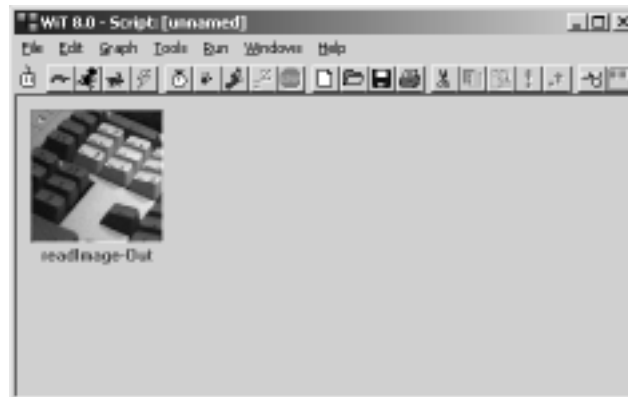
Read Image

In script mode, you invoke operators by double clicking instead of drag-and-drop. So double-click the left mouse button at **readImage** in the Operator Explorer. The operator property panel appears automatically:



readImage Operator Properties in Script Mode

Notice that this panel looks a little different than when running in igrph mode. Set **filename** to **sample.bmp**. When you click the  button, **readImage** is executed immediately (instead of being placed on an igrph), and the image appears as the first thumbnail in the workspace.



readImage Output in Script Mode

All objects are shown with a standard size cell. To look at the **sample** image at the original resolution, double click the left mouse button at the cell. The image appears in a detached window.

Notice the name **image** shown below the thumbnail. It is the name of this cell. We will need to refer to this name when we use this image later.

Specify Rectangle

Next let us define a rectangular area from the image. WiT has an operator called **getData** which allows you to interactively enter graphic objects such as rectangles or polygons on an image. We will use **getData** to create the rectangle.

Invoke the **getData** operator. Its property panel comes up. Set the **type** to **rectangle**.



getData Graphic Type

Since **getData** requires an input image, you need to tell it which object it should use as input. Click the **I/O** tab. You can type in the name of the cell produced by **readImage** as the name of **Input 0** of **getData**, but you can do the same by just point-and-click. To do that, first click the left mouse button at **Inputs 0** to make it the active field. Now click the cell produced by **readImage**. The name **image** is copied to input 0. The panel should now look like this:



getData Input Name

Click . The **sample** image appears in a separate window, with a **Graphics Editor** panel:



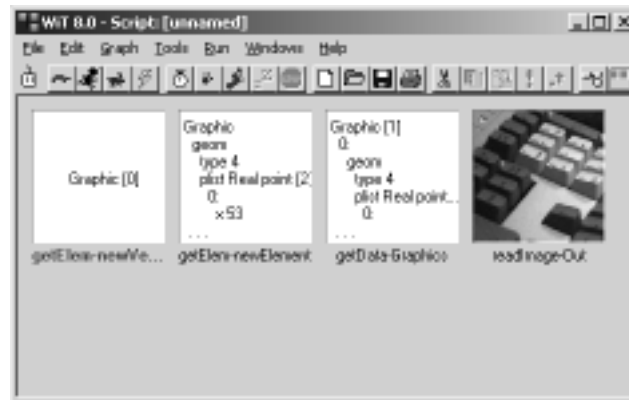
Using the getData Operator

Notice on the bottom row of the Graphics Editor, the data type is preset to  (rectangle). To enter a rectangle, press and hold the left mouse button on the image where one of the rectangle corners should be, then drag the mouse to the other corner and release the button. Bring up the property panel on the image and select the **OK** menu item. A new cell appears in the workspace.

Select Element

Now we would like to execute the **statsRoi** operator. However, **statsRoi** requires a single **graphic** object, not a vector. Even though the **getData** output is a vector of just one element, it is still a vector. To extract the single element from the vector, invoke the **getElem** operator. Set **method** to **first**, and click . Two new cells are produced. One cell is the first element, labeled **newElement** on the bottom of the card. The other is the rest of the vector, in this case an empty vector. Notice the type of **newElement** is **Graphic** (without the brackets), meaning it is a single **Graphic** object.

Because the output of **getData** is a vector of **graphic** objects, this new cell does not contain an image. Instead, the first few lines of this object are displayed. You can look at the contents of this object by clicking on the card, just as you did to display the image.



After Executing Operator getElem

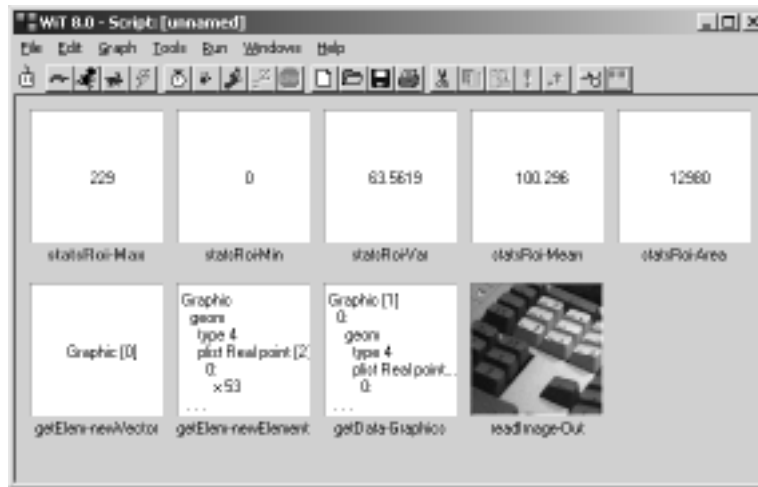
Compute Statistics

Now we are finally ready to execute **statsRoi**. It requires an image on its first input and a graphic object on its second input. Set input 0 to **image** and input 1 to **newElement**. The properties panel should look like this:



StatsRoi Operator Properties

Click . The outputs of **statsRoi** are placed in the workspace. The **mean** is the number we are looking for. Notice the cells are auto-arranged to fit in the workspace.



Statistics Output

Editing the Script

All the actions you performed are recorded in a script. To see the script, choose **Edit/Script...** from the menu bar. A script window comes up. This script can be edited and run. For example, change the **filename** for **readImage** from **sample.wit** to **brain.bmp**, and click the  button. The same script is now applied to a different image!

```
Script: d:\dev\wit\demo\eg\script.wit
readImage()
param
  filename=sample.wit,
  type=0;
output
  Out=readImage-Out
|
getData()
param
  name=getData,
  type=4,
  Placement=0,
  XScale=1,
  YScale=1;
input
  Image=readImage-Out;
output
  Graphics=getData-Graphics
|
getElem()
param
  method=0;
input
  Vector=getData-Graphics;
output
  newElement=getElem-newElement,
  newVector=getElem-newVector
|
statsRoi()
input
```

Script Window

Igraph Editing Techniques

Both operators and links are considered *igraph objects*. Many editing techniques apply to both operators and links. You have already tried the technique for bringing up a property panel. In this exercise, you will be introduced to several other useful techniques. First switch to graph mode (click ) , and clear the workspace (click ).

Selecting Igraph Objects

Create some operators and links on the igraph. To select an igraph object, click the left mouse button while pointing at it. Notice the object turns red. This is how you select a single igraph object. To perform an area select, press and hold the left mouse button at a blank location on the igraph, then drag the mouse. To add an object to the selected list, click the middle mouse button. To remove an object from the selected list, click the middle mouse button *again*.

Moving Objects

Select a few objects (both operators and links). Point the mouse at one of the selected objects, then press and hold the left mouse button. Drag the mouse. Notice all the selected objects move with the mouse. Release the mouse button. The selected objects move to the new location.

Click the  button. The objects move back to their original position. You can undo any editing action, such as copy, paste, or even link modification (see following sections).

Normally, when you move one or more operators which have links attached to them, then all the links rubber band to follow the operators. This way you can shift operators around without having to reconnect their links. If you want operators to be disconnected from unselected links when you move them, hold the SHIFT key down while you drag the mouse.

Cut, Copy, Paste, and Duplicate

Select a few objects. Then click the  button. The selected objects are deleted. Click the  button. The previously deleted objects come back. Click , then , and move the mouse inside the workspace. A copy of the selected objects now follows the mouse. Click the left mouse button to place the objects.

To perform a copy and paste in one step, select **Duplicate** from the workspace pop-up menu. A copy of the selected objects now follows the mouse. Click the left mouse button. The new set of objects is placed at where you clicked the mouse.

Editing Links

Make some links with corners on the igrph. Deselect everything by clicking the left mouse button at a blank location. To edit the link, point at a link corner, then drag the mouse with the left button down. The two line segments connected to the link rubber band to follow the mouse. Release the mouse button. The link shape is modified. Notice that the link must *not* be selected when you modify its shape. If the link is selected, then clicking the left mouse button at it will start *moving* the entire link.

Icon Appearance

From the operator property panel, you can flip an operator icon, control how each parameter is displayed, disable the operator, or bypass it. A disabled operator (or link) behaves as though it does not exist on the igrph. A bypassed operator behaves as though there are links which connect its inputs directly to its outputs. Naturally, bypass is only possible with operators which have the same number of input and output ports *and* the ports support the same data types. Examples of operators which can be bypassed are **lopass2d**, **invert**, and **median**, just to name a few. Obsolete operators are marked with a small red X in the center of the icon. All obsolete operators are contained in the obsolete folder of the operator explorer. Examples of obsolete operators are **rdObj**, **rdImage**, **startSync** and **wrObj**, just to name a few.



Disabled, Bypassed and Obsolete Operators

Zooming and Design Size

If you need more room to place operators and links, you can increase the design size. Select **Design size...** from the **Windows** menu on the WiT menu bar. A design size panel comes up:



Design Size Panel

Enter the desired width and height, then click the  button. One useful design technique is to make the design very big initially, such as 2000x2000. Then when the design is done, resize the window to be just big enough to contain the igrph, then click the  button to make the design size the same as the window size.

If your igragh gets too big to fit on the screen, you may want to zoom out (make the objects smaller) so that you can fit more operators on the screen. Point the mouse at a blank spot on the igragh design area, and press the right mouse button to bring up the property menu. Select **Zoom**. The cursor changes to a magnifying glass. Now if you move the mouse to the right and down, you will zoom in the igragh (objects become bigger). The mouse cursor changes to . If you move the mouse to the left and up, you will zoom out (objects become smaller). The mouse cursor changes to .

To pan the igragh (especially useful after you have zoomed in), use the scroll bars on the left and bottom of the igragh workspace. The scroll bars appear automatically when the design size is larger than the window size. To return to the original size, select **Normal size** from the workspace menu.

Subgraphs

When an igragh becomes too complex, portions of it can be grouped into sub-graphs to clarify the design. Sub-graphs can also be used as a way to re-use design components, much like functions or procedures in a traditional programming language. WiT displays a subgraph section like a regular operator. However, in terms of execution and implementation, the subgraph behaves exactly as if the entire collapsed section fits inside the icon that represents it.

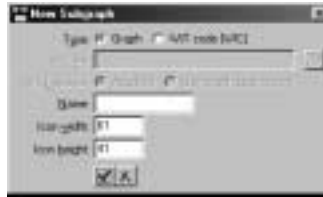
In this exercise, we will make a subgraph which scales an image to 128x128, enhance the contrast by equalizing it, and returns that as the output. Then we will make multiple instances of this subgraph to process several images within one igragh.

Adding a New Subgraph

Create three instances of **readImage** (you may want to use the **Duplicate** workspace menu item). Set the **filename** parameter of the first instance to **sample**, the second to **mri**, and the third to **fighter**. Next we need to create a subgraph that will scale an image and equalize it. Select the **Add subgraph** menu item from the **Graph** menu:



The **New Subgraph** panel comes up:



Enter **mySub** as the name, leave the **type**, **width** and **height** values at their default settings, then click the  button. A new operator appears in paste mode. Move the icon to the right of the first **readImage** operator and click the left mouse button to place it there. The new **mySub** subgraph icon is displayed with a red dot at the upper right corner to indicate that it is a subgraph icon, with no input or output ports, like this:



If you click the right mouse button on the subgraph icon, a subgraph property panel will come up:



This panel looks very much like an operator property panel, but it has 3 additional buttons:  for expanding the subgraph,  for editing, and  for editing the icon. Click the  button to edit the subgraph.

Editing a Subgraph

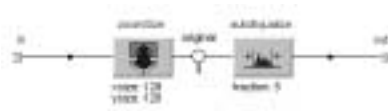
After clicking the  button on the subgraph properties panel, the workspace is cleared and you are now editing the contents of your new subgraph.

Create one instance of the **in** operator. Notice that the symbol of the **in** operator looks like an ordinary operator input port, but it is hollow and slightly larger. On the right is a solid output port, this is the output of **in**. Create one instance of the **out** operator. The solid input port on the left is the input port of **out**.

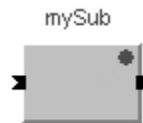
Create one instance of **zoomSize**. Set both the **xsize** and **ysize** parameters to 128. Then create one instance of **autoEqualize**. Leave the fraction parameter at the default value of 5.

Connect the output port of **in** to the input of **zoomSize**, the output of **zoomSize** to the input of **autoEqualize**, and connect the output of **autoEqualize** to the input of **out**.

You should now have an igragh like the following:



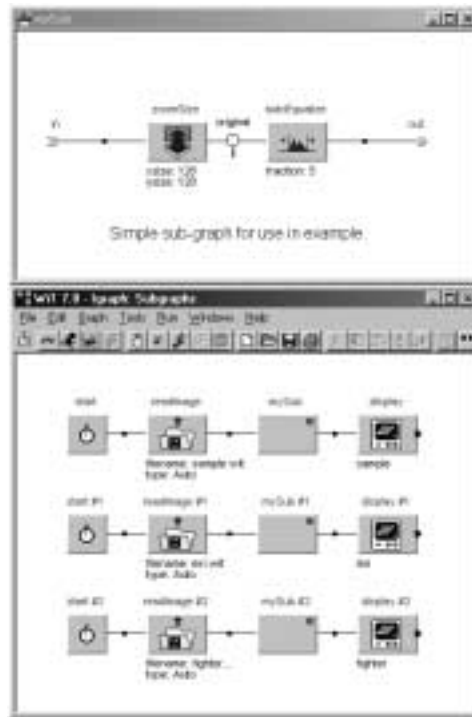
Click the  button on the tool bar to go back up to the top-level igragh. You should now see an input port and an output port on the **mySub** icon, like this:



Select the **mySub** icon, then choose **Duplicate** to make another copy. Repeat to get a total of three copies of **mySub** on the workspace. Connect the output of each of the **readImage** operators to an instance of **mySub**. Create three instances of **display**. Set the **name** parameter of the first instance to **sample**, the second to **mri**, and the third to **fighter**. Connect the output of each of the **mySub** operators to an instance of **display**.

Running an Igragh with Subgraphs

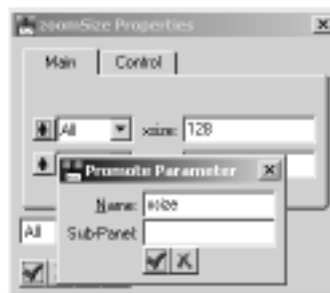
Bring up the properties panel for any one of the **mySub** icons and click the  button. You will see the igragh that the **mySub** icon represents come up in a pop-up window:



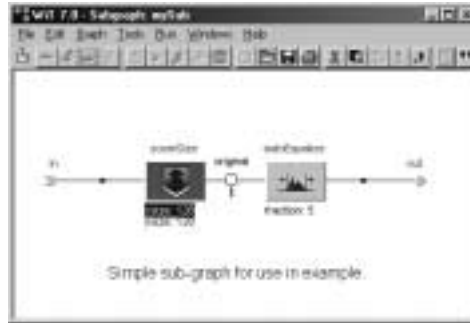
Run the igrph while the **mySub** graph is visible. You will see that visual feedback (operators turning green, token travel, etc.) applies to displayed subgraphs also.

Promoting Parameters from a Subgraph

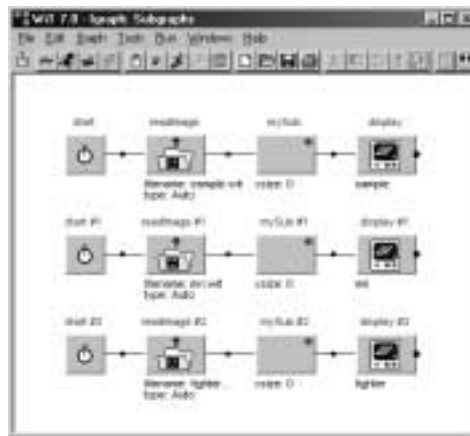
You can modify the behavior of a subgraph by promoting some of the parameters of its operators. For example, bring up the properties panel for one of the **mySub** icons and click the  button to edit the subgraph. Bring up the properties panel of **zoomSize**, and click the  button next to the **xsize** parameter. The **Promote Parameter** dialog comes up:



Click the  button to accept the default name. Notice that the **xsize** parameter is shown in reverse video to indicate that it has been promoted:



Click the  button to go back to the top level. Now you should see a new parameter named **xsize** appearing under all instances of the **mySub** icon. You can now change the **xsize** parameter without editing the subgraph.



Learning More

You should now have a fairly good feel for what WiT can do. To learn more about WiT, read the *User's Manual*. You should also go through all the other demo igraps in the **wit\demo** directory. The **Eg** directory contains the examples in the tutorial that you have already gone through. The **Operators** directory contains interesting demos for many of the WiT operators. The **Methods** directory contains more application (rather than operator) oriented demos. These demos tend to be the most complex.

The best way to learn WiT is to start using it for real problems. Start with some simple ones. When you want to do something that you don't quite know how to do, try to remember if you have seen something similar in one of the demos. For example, the demos contain examples of how to process a list of images, running multiple operators in parallel or in a pipeline, fill holes in binary images, etc. Go through the *Operator Reference Manual* to see what the standard WiT operators can do. If you are a good programmer, read the *Programmer's Manual* to learn how you can create custom applications in C/C++ or Visual Basic, or add new operators in C/C++. The *Reference Manual* has 'User' and 'Programmer' sections with reference details that correspond to the *User's Manual* and *Programmer's Manual* respectively.

If you are really stuck, try our web site **www.wit-igraph.com**. If you still cannot find the information you need, email us at **wit@corecoimaging.com**.

Coreco Imaging Contact Information

Sales Information

Web site: www.imaging.com
Email: info@corecoimaging.com

Technical Support

Voice: (604) 435-2587 ext 9
Email: wit@corecoimaging.com

Corporate Headquarters

Coreco Imaging Inc.
7075 Place Robert-Joncas, Suite 142
St. Laurent
Quebec H4M 2Z2
Canada

Tel: (514) 333-1301
Fax: (514) 333-1388

US Sales Office

Coreco Imaging Inc.
900 Middlesex Turnpike
Building 8, Floor #2
Billerica, MA 01821
USA

Tel: (781) 275-2700
Fax: (781) 275-9590

